L^{e}A
Id.: USER
Version: 3.5
Date: July 18, 2002
Revised: July 18, 2002

GSI Lean Analysis, User Manual

H.G. Essel

July 18, 2002

GSI, Gesellschaft für Schwerionenforschung mbH
Planckstraße 1, D-64291 Darmstadt
Germany
Tel. (0 6159) 71–0

List of Figures

1.1 Scattered L[e]A Logo 3

2.1 Scatterplot with Polygon Condition 9
2.2 Histogram Definition Menu 11
2.3 Command Definition Menu 19

3.1 Main Control Menu 27
3.2 Output menu and rate meter 28
3.3 Display 30
3.4 Display Buttons 30
3.5 Smart Expander 31
3.6 Pictures 32
3.7 2D Control Menu 33
3.8 Lettering 34
3.9 Polygon Editor 35
3.10 Condition Menu 36
3.11 Polygon Condition Editor 37
3.12 Fit Menu and Result 38
3.13 Histogram Header 39
3.14 Histogram menues 40

Chapter 1

LEA Introduction

GM_LEA_USER GMC_LEA_INTRO

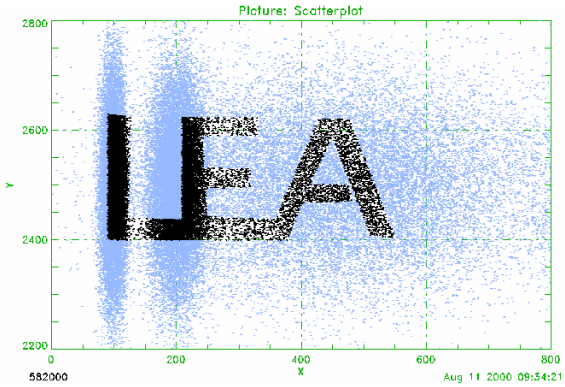

Figure 1.1: L^eA as polygon condition

1.1 Information about L^eA

1.1.1 Web Site

More information about L^eA , e.g. manuals and publications can be found in the Web under

`http://lea.gsi.de`

1.1.2 Mailing List

One may subscribe to the L^eA mailing list:

`DAQ-LEA@GSI.DE`

1.1.3 LeA News

In the lealogin profiles LeA news are announced. To read the news, use `cnews` on OpenVMS, and `new` on Unix:

```
cnews lea -a    ! list of all news
cnews lea -u    ! display all unseen news
cnews lea 15    ! display news item number 15
```

Only unseen news are announced.

1.2 Functionality

LeA runs currently under OpenVMS, Linux, and AIX. The visualisation is done by IDL (*Interactive Data Language*) developed by Research Systems Inc. A developer license is required. The software is written entirely in C and IDL. LeA is multi-threaded, i.e. parallel I/O, graphics and command execution is possible. There are four ways to use LeA :

1. Use prebuilt LeA programs filling histograms.
2. Use prebuilt LeA programs for histogram display (histograms from server)
3. Provide an analysis function which is linked into standard LeA .
4. Write explicitly the event loop (no graphics!).

The last method is described in section 2.5, page 22.

1.2.1 LeA in MBS

LeA without graphics runs also on MBS (see 2.6, page 22). In this case the analysis function is called event by event in the MBS collector, and may change the event or skip further processing (software trigger). The on-line histograms can be accessed via the MBS histogram server from a remote LeA , from Microcal Origin or the Go4viewer. A histogram C-API is provided to receive histograms in any program.

1.2.2 User Interface

LeA uses the same user interface as MBS , the standard data acquisition system at GSI. This interface can run in line mode or in menu mode.

Line Mode

When running in line mode typing one ? one gets the list of commands, two ? print the commands with their arguments. Additional arguments select commands. Typing a command (with all command keywords) followed by one ? one gets the argument description, i.e.:

```
lea> ??
lea> ?? cre his
lea> plot hist ?
```

You leave the program by command `quit`.

Window Modes

The user interface provides two window modes: One is for graphics only, one for full LeA control and graphics. The graphics window can run synchronously or asynchronously. When running synchronously, the command line input is blocked, but graphics output from the analysis routine is possible, i.e. for scatterplots. When running asynchronously, or with the control window, no graphics output is possible in the analysis routine. Therefore the usage of the control menu is recommended when no scatterplots are needed, whereas the command line interface is recommended for batch.

User Specific Commands

A user may define additional commands (in initialization part of the analysis function) and additional GUI components from where these or standard commands can be executed.

1.2.3 Histogramming

The standard part of the analysis program LeA includes the handling of histograms, like create, show, clear, delete, fit, calculate, display etc. Histograms can be dumped in standard GSI format (gef) to text files and processed by GOOSY, SATANGD or PAW. Histograms can also be restored from these files. The histograms are organized in a piece of memory called base. A complete base can be dumped and restored. Currently the base is located in local memory and therefore lost on program exit except it is dumped. A complete base from LeA running inside MBS (Lynx) can be restored on any LeA platform.

1.2.4 Conditions

LeA supports window (2 dimensional) and polygon (up to 32 points) conditions. The conditions can be set (graphically or by values), dumped to text files and restored. A C-API is implemented including macros for checking conditions in the user analysis function. The conditions are located in the data base and dumped/restored with the base.

1.2.5 Data I/O

Data input/output is provided online from data acquisition system MBS (synchronously from transport or sampling from stream server or from event server), from remote event servers and

from and to files. There are commands provided to inspect and copy raw data files formatted according the GSI event data format:

Type Events

The `type events` command reads events from several input sources: MBS transport, MBS stream server, MBS event server, remote event server, or file, and prints formatted data. Events can be skipped.

Type Buffer

The `type buffer` command is similar to `type event` but works on buffers. Buffers can be skipped. Input only from MBS transport, MBS stream server, or file. This command works similar as the MBS `type event` command.

Copy Events

The `copy events` command copies events from the input sources into standard GSI formatted raw data files.

Copy Buffers

The `copy buffers` command copies buffers from the input sources into raw data files. Optionally data is compressed. Compressed input data is uncompressed. Without output file this command can be used to check buffer consistency.

Analyze Events

The `analyze events` command gets events from the input sources and calls the user analysis routine `f_anal` or `f_mbs_anal`. Optionally this routine may return new events, which are then written as GSI formatted raw data files.

Type File

The `type file` command dumps files similar to the VMS `DUMP` command.

Chapter 2

Getting Started

GMC_LEA_START

2.1 Preliminaries

Before using L^{e}A some definitions have to be done by calling

```
VMS:          lealogin [v2] [v3]
AIX, Linux: . toollogin
            . idllogin
            . lealogin [v2] [v3]
```

That could be done in your `LOGIN.COM` or in `.profile`, respectively. Version 3 is enabled by `v3`.

2.1.1 Help

Help can be obtained in the menues or by command `leahelp`. On VMS, the standard help mechanism is used, on Unix something similar. The difference is that on Unix the help command always returnes:

```
> leahelp
> leahelp lea_intro
> leahelp lea_intro examples
```

2.2 Standard L^{e}A Programs

2.2.1 Histogramming

If you want to have a look to L^{e}A and just play you may run one of the two ready to use L^{e}A programs. They accumulate histograms from two kinds of events: `lea -tab -men` assumes

events written by MBS table readout, and `lea -fun -men` assumes events written by an MBS user readout function with just longwords for values. Both programs check if there are one or two subevents, and fill up to eight histograms (4096 channels) per subevent. They run in the same way on Linux, OpenVMS and AIX.

The main difference between OpenVMS and Unix is now the source directory of the sample data file (`leadir` is the value of `LEA_DIR`).

Samples

```
$ lea -fun
lea> anal ev lea$src:gauss -all            # VMS
lea> anal ev <leadir>/examples/gauss -all  # Unix
lea> disp -start
lea> plot histogram se1_1 color=0 -scale -fit
lea> disp
```

Command `disp -start` starts the display, command `disp` starts the display menu. The command line is blocked until you leave the menu.

2.2.2 Special Features

The `-fun` version provides three special features:

Scatter Plot

With the `set scatter` command one can specify a scatterplot. Note, that scatter plots work only from the command line without menu.

```
set scatter subevent=lo channel=lo color=lo -X -Y -ON -OFF
subevent : select subevent ID
channel  : select channel offset in this subevent (0,1,2,...)
color    : select color of points (1-15)
-X       : specified channel is X
-Y       : specified channel is Y
-ON      : enable scattering
-OFF     : disable scattering
```

Assume you have two subevents with ID 0 and 3 (specified in the MBS setup file) and you want scatter the first longword of subevent ID=0 to second longword of subevent ID=3.

```
lea -fun
lea> disp -start
lea> scatter 0 4000 0 4000 chan_0_1 chan_3_2 title
lea> set scat 0 0 -X
```

```
lea> set scat 3 1 -Y
lea> set scat col=4 -ON
lea> anal event ...
```

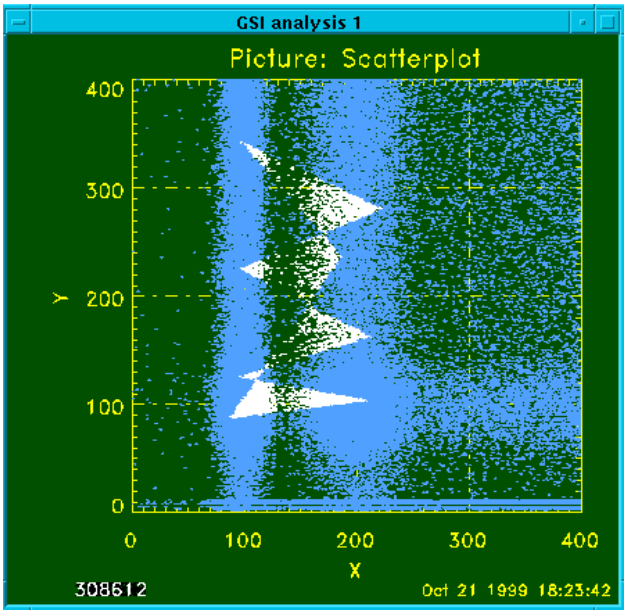

Figure 2.1: Scatterplot with Polygon Condition

The scatter picture can be saved in different formats on disk:

```
lea> plot image <file> -GIF -BMP -PS -TIFF -JPEG
```

Depending on the type `.gif .tif .bmp .jpeg .ps` will be appended to the file name. `-TIFF` and `-JPEG` currently do not yet work.

Polygon Condition

To get used with polygones, one may activate a polygon by command

```
lea> polygon <file> -on
```

The polygon `file` must be created first as described in chapter 3.2.8, page 34. If you display a scatter plot with activated polygon, you will see the points inside the polygon in a different color as shown in Fig. 2.1. If you accumulate histograms, the 2D histogram will be filled only inside the polygon.

Plot Polygon

A polygon is plotted in the scatterplot frame by

```
lea> plot polygon <file> color=c
```

Tuple Mode

With the command `set mode -TUPLE` one can select a simple ntuple mode (`set mode -HISTO` switches back). In this mode, 8 channels per subevent are stored value by value in the histograms. To get a scatter plot, display the first and the second histogram (channel) and press then the `Corr` button.

2.2.3 Histogram Client

To use LeA as graphic tool for remote histogram servers like MBS or GOOSY, call

```
lea -rem <node> <base> [-a=<password> -p=<port>]
lea -cop <node> <base> [-a=<password> -p=<port>]
```

The first call works with any GSI histogram server. It gets the base structure from the server, creates the histograms and copies the contents. The second call works with MBS servers. It copies the whole base at once. See also 3.3.5, page 40.

2.3 Writing an Analysis

All histograms and conditions used in the analysis routine must be created after startup. This is done in a startup profile. Furthermore the histogram pointers must be defined and set in the analysis routine, which is done by two include files. To keep these files consistent one can create and modify the histogram definitions through a menu. All other files needed are then created automatically. All file names have one name in common, the *project* name. In the following we use in the examples the project name **mylea**.

2.3.1 Create Project Files

Start a new project by

```
lea -new <project>
```

where <project> is your new project name (alphanumeric, three to six chars). The definitions should be modified only by command `lea -mod` <`project`> which starts a histogram definition menu (Fig. 2.2). On the left side one specifies the histogramm parameters. With the `Apply` button the histogram is entered into the list. To modify a histogram, click on the name in the list, change values, and `Apply` again. When all histograms are ready, first `Save` the setup. The default file name is <`project`>`.lea`. Then press `definition file` which writes a file <`project`>`.def`. Then `Quit` and all other files are created.

The following files should **not** be modified except by command `lea -mod mylea`:

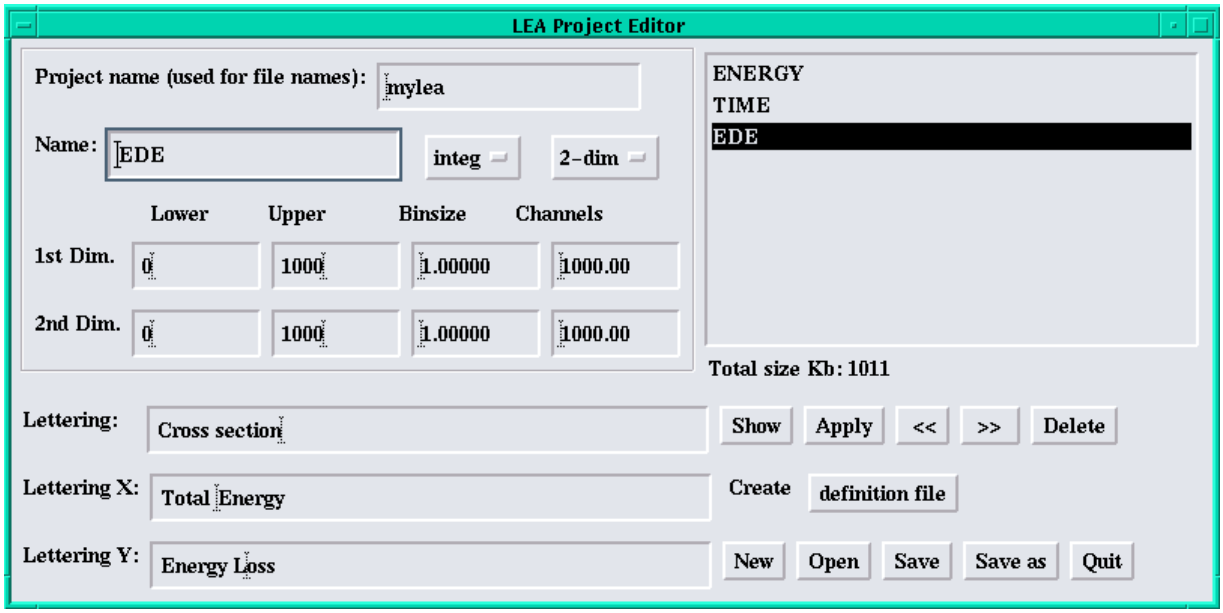

Figure 2.2: Histogram Definition Menu

```
mylea.lea           !IDL file with histogram definitions
mylea.def           !ascii file with histogram definitions
hisdef_mylea.h      !include histogram pointer definitions
hisini_mylea.h      !include histogram pointer setting
hiscre_mylea.scom   !histogram creation for startup
```

The following files can be modified:

```
f_anal_mylea.c      !analysis function
f_mbs_anal_mylea.c  !analysis function to run also in MBS
bascre_mylea.scom   !data base creation for startup
leaini_mylea.scom   !startup creating a base (calling bascre_mylea and hiscre_mylea)
leaini_mylea.idl    !IDL profile
Makefile.mylea      !Makefile (gmake)
Makefile_mbs.mylea  !Makefile for mbs
```

The following files can be created optionally:

```
leaexit_mylea.scom  !commands executed before exit, e.g. dump base
```

The analysis function is a project specific working noop. The generated analysis program can be started by

```
lea -pro mylea -men
```

2.3.2 Develop on OpenVMS

The following commands may be abbreviated:

```
leacomp   file=?  /DEB*UG/LIB*RARY=<object library>/INC*LUDE=<include directory>
lealink   files=? /DEB*UG/LIB*RARY=<object library>/EXE=<executable file name>
lea -make mylea
```

The last line simply combines `leacomp` and `lealink` assuming the standard name for the analysis function (`f_anal_<project>.c`). If one needs more files to compile and to link, the explicit commands must be used. Note that the optional object library is output for `leacomp` but input for `lealink`. Example: assume in `f_anal_mylea.c` is your analysis routine calling another routine `f_anal_calc`:

```
$ leacomp f_anal_calc /lib=mylib.olb
$ leacomp f_anal_mylea
$ lealink f_anal_mylea /exe=m_lea_mylea /lib=mylib.olb
$ lealink f_anal_mylea,f_anal_calc /exe=m_lea_mylea
```

The last command links `f_anal_calc` from the file instead from the library. If you do not specify a name for the executable, `m_lea.exe` is created.

2.3.3 Develop on AIX and Linux

On the Unix platforms one should use the makefile `Makefile.<project>` created with the user function. Note that you execute it by one of these two commands:

```
lea -mak mylea
gmake -f Makefile.mylea
```

2.3.4 The Analysis Function (f_anal)

In your analysis file there are at least two functions: `f_anal_init` and `f_anal`. The first one is called twice during startup (see 2.4.2, page 21), the second for each event. Command `analyze events` starts reading events from the specified input source and calls the user routine `f_anal` with the arguments:

```
l_status = f_anal(ps_evhe,ps_bufhe,ps_filhe,c_in,*ps_evhe)
ps_evhe : (s_evhe *) pointer to event If this pointer is zero, input stream was closed.
ps_bufhe: (s_bufhe *) pointer to buffer header.
ps_filhe: (s_filhe *) pointer to file header, if present.
pc_input: (CHARS *) pointer to input
*ps_evhe: (int) return address of output event.
```

When the input finished, `f_anal` is called once more with an event NULL pointer. Checking this pointer, the routine can do some cleanup etc. When the routine returns status `ANAL_STOP`, input is stopped and `f_anal` is called once more with an event NULL pointer. The user routine may calculate a new event and return a pointer to this event to the analysis program. This event will be written to an output file when

1. the file had been opened by the `analyze event` command,
2. the return status is not `ANAL_SKIP`,
3. the returned pointer is not NULL.

The output event can be of type/subtype = 10,1 structure as the MBS events, or of type/subtype 4,1. The format of such events must be a longword with the data size in 16-bit words followed by two words type/subtype (4,1), and then the data. The buffer type/subtype must be specified as 4,1 in the `analyze event` command.

Histogram Access

To access, i.e. accumulate, histograms in the analysis routine, pointers for each histogram must be declared and initialized. This is done in the include files described above. The generated analysis template includes these files at the right places.
For histogram accumulation macros are provided:

Macro HISACCnt

The `n` can be `1` or `2`, and `t` stands for `I` (integer histogram) or `R` (floating point histogram). This macro can be called in `f_anal` to accumulate a histogram. The arguments are the histogram name, the channel(s), and the increment, both `float`. Example:

```
HISACC1I(SP1,r_x,1.);
HISACC1R(SP2,r_x,1.);
HISACC2I(SP1,r_x,r_y,1.);
HISACC2R(SP2,r_x,r_y,1.);
```

Macro HISSETnt

These macros have different arguments in that the channel is the bin number. Instead of incrementing the channel content they set the channel content.

```
HISSET1I(SP1,l_bin,5000.);
HISSET2R(SP2,l_bin_x,l_bin_y,284.);
```

Macro HISCLRt

These macros clear a histogram.

```
HISCLRI(SP1);
HISCLRR(SP2);
```

Sometimes one wants to have direct access to the histograms. There are two static pointers defined and set for each histogram, one for the header, one for the data. The header pointer is `ps_spe_NAME`, the data pointer `pl_NAME` or `pr_NAME`, depending on the data type. `NAME` is the histogram name in UPPERCASE. The header stucture is `s_spe.h` on `LEA$INC` or `$LEA_DIR/inc`.

Histogram Arrays

Sometimes it is convenient to have indexed histograms, i.e. to accumulate histograms in a loop. But LeA does not support histogram arrays. There are two work arounds.

The first one is to use the rows of a 2-dim. histogram as 1-dim. histograms. The histogram is filled with an index as second argument (Y). The GUI supports the selection of histogram rows by a slider below the histogram list (Fig. 3.1, page 27). A value of n > 0 means to process the nth row as 1-dim. histogram, 0 means to process 2-dim histogram.

The other work around requires more programming, but can also be used for 2-dim. histograms. Define static pointer arrays for histogram headers and data. Behind the `#include "hisini_...h"` copy the pointers of histograms to the arrays. In the event loop use functions

```
f_his_accu1(s_spe *, INTS4 *, REAL4, REAL4)
f_his_accu2(s_spe *, INTS4 *, REAL4, REAL4, REAL4)
```

instead of the macros. First and second argument are the pointers to the histogram header and the data, respectively. Example:

```
/* before f_anal_init */
s_spe *ps_spe_adc[8];
INTS4 *pl_dat_adc[8];
....
/* this include is already there */
/* histograms S1...S7 exist      */
#include "hisini_adc.h"
  ps_spe_adc[0]=ps_spe_S1;
  pl_dat_adc[0]=pl_S1;
...
  ps_spe_adc[7]=ps_spe_S7;
  pl_dat_adc[7]=pl_S7;
...
/* in the event loop, r_adc[i] may be a data array */
for(i=0;i<8;i++) f_his_accu1(ps_spe_adc[i],pl_dat_adc[i],r_adc[i],1.0);
/* accumulates eight values r_adc[i] into histograms S1...S8 */
```

2.3.5 Conditions

Similar to the histograms, all conditions must be created during startup. The values of window limits or polygon points can be edited later, but the names must be known. Each condition has

a test and a true counter and can be freezed to true or false. The conditions are stored in the data base.

Window Conditions

Typically conditions are created in a LeA procedure which is called in the startup procedure before the `init` command, e.g. in `leaini_mylea.scom`. The procedure could look like this:

```
create directory window 10
create window w1 "first description string" 10 20 10 20
create window w2 "2nd description string"
set wind w2 10 20 10 20
restore window wincon.win -over
```

The first command allocates 10 window conditions. Then two windows are created (just to demonstrate), which are then overwritten by the windows previously dumped in file `wincond.win`. Optionally limits can be specified. The default limits always result in FALSE. All window conditions keep two pairs of limits. Alternatively only one or both limit pairs can be checked.

Polygon Conditions

Polygon conditions can have a maximum of 32 points. The polygons can be edited graphically. Creating polygon conditions is similar to creating window conditions:

```
create directory polygon 10
create polygon p1 "first description string"
create polygon p2 "2nd description string" poly.pold
restore polygon polycond.pol -over
```

The first command allocates 10 polygon conditions. Then two polygons are created (just to demonstrate), which are then overwritten by the polygons previously dumped in file `polycond.pol`. In the second `create` command a polygon data file `poly.pold` is specified. Such text files (created by the polygon editor, or by command `export polygon data`) keep the points for one polygon as opposed to `polycond.pol` (created by the condition editor or command `dump polygon`) which keeps **all** polygon conditions.

Conditions Checking

Conditions are checked by calling functions or macros. Both use the condition index which is returned by the functions:

```
index=f_win_loc("w1");
index=f_pol_loc("p1");
```

These functions must be called in the initalization part (command `initialize`).
Conditions are then checked by

```
result=f_win_1(index , x);
result=f_win_2(index , x, y);
result=f_poly_check(index , x, y);
```

Because the index is simply defined by the order of creation starting with zero, one can execute the functions in loops, e.g. together with indexed histogram pointers as described in the previous section.

Condition Macros

Less flexible, but easier is the usage of macros. Start any LeA and create all window and polygon conditions you need. Then execute

```
create file mylea -window
create file mylea -polygon
```

These commands create the files

```
poldef_mylea.h
windef_mylea.h
```

Include these files in the `initialize` command of your analysis function. They define for all condition indices names and check at runtime if these names are the correct indices. In the event loop conditions are checked by

```
if(WINCOND1(w1,x))   { ... }
if(WINCOND2(w2,x,y)) { ... }
if(POLYCOND(p1,x,y)) { ... }
```

Note that here similar to the accumulation macros the condition names are **not** strings, and they are **lowercase**.

2.3.6 Event Structures

Byte Order

Big Endian: least significant bit is in byte with highest address (6800, IBM).
Little Endian: least significant bit is in byte with lowest address (Intel, Alpha).
Between machines with different byte ordering, the bytes in a longword must be swapped and all word or byte declarations in mapping structures must be declared in reversed order, e.g.:

```
short i_subtype;
short i_type;
char  h_begin;
char  h_end;
short i_used;
```

must be declared on a machine with different endian as:

```
short i_type;
short i_subtype;
short i_used;
char  h_end;
char  h_begin;
```

In L[e]A all input is swapped if necessary, structures are defined correctly, and the event structure members can be referenced independent of the byte ordering.

Event

Little Endian representation:

```
typedef struct
{
long  l_dlen;     /*  Data length + 4 in words        */
short i_type;     /*  Type number                     */
short i_subtype;  /*  Subtype                         */
short i_dummy;    /*  Not used yet                    */
short i_trigger;  /*  Trigger number                  */
long  l_count;    /*  Current event number            */
} s_vehe;
```

Subevent

Little Endian representation:

```
typedef struct
{
long l_dlen;      /*  Data length +2 in words         */
short i_type;     /*  Type number                     */
short i_subtype;  /*  Subtype                         */
short i_procid;   /*  Processor ID [from setup]       */
char h_subcrate;  /*  Subcrate number                 */
char h_control;   /*  Processor type code             */
} s_veshe;
```

For table readout the data per CAMAC channel is

```
typedef struct
{
short i_index;    /*  CAMAC parameter table index     */
short i_data;     /*  CAMAC data value                */
} s_data;
```

When using the standard readout mechanism (readout tables), the CAMAC data follows as 1 word (16bit) channel data and 1 word channel number. Channels with a data value of zero are suppressed.

GSI File Header Buffer

Little Endian representation:

```
struct cv_string
{
    short string_l;
    char string[78];
} ;
struct s_filhe
{
long filhe_dlen;        /* length of data in words         */
short filhe_type;        /* = 2000                          */
short filhe_subtype;     /* = 1                             */
short filhe_used;
short filhe_frag;
long  filhe_buf;
long  filhe_evt;
long  filhe_current_i;
long  filhe_stime[2];
long  filhe_free[4];     /* free[0] = 1 -> swap             */
short filhe_label_l;     /* length of tape label string     */
char  filhe_label[30];   /* tape label                      */
short filhe_file_l;      /* length of file name             */
char  filhe_file[86];    /* file name                       */
short filhe_user_l;      /* length of user name             */
char  filhe_user[30];    /* user name                       */
char  filhe_time[24];    /* date and time string            */
short filhe_run_l;       /* length of run id                */
char  filhe_run[66];     /* run id                          */
short filhe_exp_l;       /* length of explanation           */
char  filhe_exp[66];     /* explanation                     */
long  filhe_lines;       /* # of comment lines              */
struct cv_string s_strings[30]; /* max 30 comment lines     */
} ;
```

GSI Buffer

Little Endian representation:

```
typedef struct
{
long  l_dlen;      /*  Length of data field in words       */
short i_type;
short i_subtype;
short i_used;      /*  Used length of data field in words  */
char  h_end;       /*  Fragment end at begin of buffer     */
char  h_begin;     /*  Fragment begin at end of buffer     */
long  l_buf;       /*  Current buffer number               */
long  l_evt;       /*  Number of fragments                 */
long  l_current_i; /*  Index, temporarily used             */
long  l_time[2];   /*  Time stamp                          */
long  l_free_0;    /*  Byte order longword = 1             */
long  l_free_1;    /*  Length of last event in buffer      */
long  l_free_2;    /*  = 0                                 */
long  l_free_3;    /*  = 0                                 */
                   /*  data field                          */
} s_bufhe;
```

2.3.7 Creating Commands

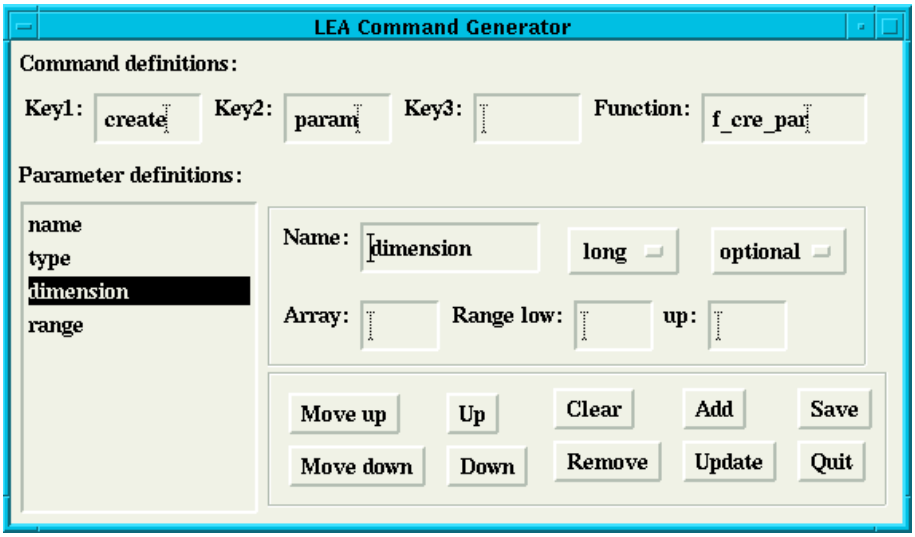

Figure 2.3: Command Definition Menu

In the entry `f_anal_init` one may define more commands. For convenience, one can use a menu (Fig. 2.3, page 19) to specify all necessary information and to get the definition call and

the action routine. First call LeA :

```
> lea -men
```

The menu is now called by `File->Command`. Enter up to three command keywords and a function name. Then specify the arguments. The `Array` specification is normally needed only for strings. When the argument specification is OK, press `Add`. To change a specification, click on the name, modify values and dont forget to press `Update`. To generate the file <function>.c, press `Save`. The file cannot be modified through the menu. The file is executable by

```
leacomp <function>
leamlink <function>
./<function>
>>> ??
```

prints the command definition. The file can thus be used to test the command definition(s) after modifications.

You must copy the command definition call from the file behind the definition of the `initialize` command in your analysis routine, and copy the action routine behind `f_anal`.

2.4 Starting a LeA Program

The standard and recommended way to start LeA is to use script `lea`:

```
lea
lea ?
lea -tab [-disp | -men]
lea -fun [-disp | -men]
lea -new <project>
lea -mod <project>
lea -pro <project> [-disp | -men]
lea m_lea_mylea [-disp | -men]
```

The last line calls program `m_lea_mylea`. Calling your program through the script, there are little differences:

```
Unix: lea m_lea_mylea [-disp | -men]
                      [-start=[<scom>]] [-x=[<scom>]]
                      [-idl=[<idl>]]
VMS:  lea m_lea_mylea [-disp | -men]
                      [-start=<scom> | -nostart] [-x=[<scom>]]
                      [-idl=<idl> | -noidl]
```

where <scom> is the command profile, <idl> the idl profile. The difference is that on UNIX with `-s=` and `-i=` NO profile is used, whereas on VMS `-nostart` and `-noidl` must be used for this case. With `-x=` an exit command file will be executed.

2.4.1 Include/Exclude Histograms from GUI

When starting the menu from command line (the normal way), one can include and/or exclude histograms from the GUI histogram list:

```
lea -pro mylea -men -l=list -e=list
```

The wildcard from -l is used to select histograms, from -e to exclude histograms, i.e.:

```
lea -pro mylea -men -l=d* -e=*x*
```

selects all histograms beginning with D but excludes all of these containing an X.

2.4.2 LeA Startup Steps

Note that your generated `f_anal` function is already complete! When you execute your LeA program the following steps are performed:

f_anal First Initialization

The user supplied initialization entry `f_anal_init` is called the first time with a NULL pointer as argument. In this case, `f_anal_init` must define the command `initialize` with execution entry `f_anal_init`. The arguments can be defined as needed. More commands can also be defined here.

Start Procedure

Then the command profile is executed, normally the generated `leaini_<project>.scom`. This procedure is used to create the histogram base and all histograms and conditions needed.

f_anal Second Initialization

The analysis is initialized a second time, but now by the `initialize` command mentioned above. Thus `f_anal_init` is called a second time by command `initialize`, but now the argument is a pointer to the argument block. In this case the include file `hisini_<project>.h` sets the histogram pointers (see below). Other setups can be inserted here, e.g. the condition includes. If an MBS function (`f_mbs_anal`) is used, `f_mbs_anal_init` is called.

Menues

Optionally the display or control menu is started, when `-disp` or `-menu` as specified, respectively.

2.5 Event Loop

The same profiles and analysis function can be used in a program which implements the event loop explicitly using the event IO routines. A functional example is `m_lea_user.c` on `lea$src` (VMS) or `$LEA_DIR/src` on unix, respectively. In such programs no graphics can be called. But it can be useful either to generate new event files or to produce histograms.

2.6 LeA in MBS

A LeA analysis can also run inside MBS . The same project files as for standalone LeA are needed. The analysis function `f_mbs_anal` gets as argument a structure:

```
l_status=f_mbs_anal(s_form * ps_form)
  ps_form->bh_trig_typ    : trigger type of current event
  ps_form->bh_n_frag      : number of subevents
 *ps_form->pl_dat_start[i]: pointer to subevent data
  ps_form->bl_dat_len[i]  : length of subevent in bytes
  ps_form->l_byte_order[i]: byte order scheme of this source
```

One can link an MBS analysis function also with standalone LeA , because the default `f_anal` fills the above structure and calls `f_mbs_anal`.

2.6.1 Writing an Analysis in MBS

1. Make a new directory (Lynx)
2. Make a histogram definition file <`project`>`.def` on Linux, AIX, or VMS as described above and ftp to the new directory.
If you create a project on Linux, AIX, or VMS, the analysis function and makefile for Lynx is already created. You may ftp these directly to your Lynx directory, or execute step 3. 3. Execute command (Lynx):

```
> histogram create files <project> -anal
```

4. Rename the generated `Makefile.<project>` to `Makefile`
5. Execute make
Now you have a runnable `m_collector` linked with your `f_mbs_anal` analysis function. Now edit `f_mbs_anal_`<`project`>`.c`, insert your analysis, and make again.
An example is `f_mbs_anal_adc.c` on `$MBSROOT/exa/analysis`. You need a histogram data base. The base is implemented as shared segment, i.e. it exists independent of running programs until reboot. The LynxOS `smems` command will show it. This implementation is different from standalone LeA . The base is created by

```
> histogram
histogram> @bascre_<project>
histogram> @hiscre_<project>
histogram> sho histo
histogram> quit
```

In your `startup.scom`, insert (assuming the base exists)

```
start task ./m_collector
start task m_histogram
attach base <project>
```

When MBS is running, switch on/off histogramming by command:

```
enable histo
disable histo
```

The first command triggers a call to `f_mbs_anal_init`. The histogram base can be dumped/restored, copied to another platform and restored by standalone LeA . Origin (WNT), LeA , or Go4viewer can be used to display MBS histograms (see 2.2.3, page 10 and 3.3.5, page 40);

Chapter 3

Operate LEA

GMC_LEA_OPER

3.1 Menu Interfaces

As mentioned above, the user interface provides two window modes: One is for graphics only, one for full LeA control and graphics. The graphics window can run synchronously or asynchronously. A description of the menues can be obtained by

```
leahelp dsp_menu
leahelp lea_control
```

3.1.1 Graphics Synchronous Menu

This menu is started in line mode. Scatterplots and plot commands are possible.

```
lea -pro mylea
lea> display
```

Command `display` starts the display using the IDL graphics. It has the following optional arguments:

```
command   : Optional command string executed in IDL after init.
histogram : Optional wildcard specification for histograms to be exported to IDL.
exclude   : Optional wildcard specification for histograms excluded from IDL.
-START    : Start IDL
-STOP     : Stop IDL
-MENU     : Start IDL menu.
-LOOP     : Enter IDL command input, leave with 'q'.
```

Called without arguments, IDL is started, then a display menu window is popped up. The Menu can be left any time (Button `File->Quit`), and called again. The <`Help`> buttons give hints for using the display. Note that you must leave the display menu to execute L^{e}A commands.

3.1.2 User Graphics

When started in line mode, several graphic levels are provided:

Interactive

Once the display has been started, one may enter IDL input mode by L^{e}A command `display -loop`. The prompt changes to `idl>` and IDL commands can be entered. Leave IDL by `quit`. You may execute your own IDL procedures. To access histograms in IDL you must get a copy. This is achieved by IDL function `LEAGETHIS("NAME")` which returns the data array of the specified histogram. Example:

```
lea> disp -l
idl> x=leagethis("SE1_1")
idl> plot,x,color=2
idl> q
lea>
```

Note that histogram names are all uppercase.

Callable

You may execute IDL commands from your routines by calling `f_his_idl_exec(string)`, where `string` is an IDL command. This works only without menus!

User Control Menues

It is possible to add a user control menu to the L^{e}A control menu. The menu must be in file `leauser.pro`. It is called through `options->LEAuser`. An example is `LEA$LIB:LEAUSER.PRO ($LEA_DIR/src/idl/leauser.pro)`. To make your `leauser.pro` accessible from other directories than the current, add these directories to the IDL searching path. This is done in your IDL profile `leaini_<project>.idl`:

```
if !version.os eq "vms" then !path = 'mydir,'+!path
if !version.os ne "vms" then !path = 'mydir:'+!path
```

The current directory is searched always first. Note that in IDL procedures no `if-else` struct is possible.

Exeuting IDL Commands

It is possible to execute IDL commands from the main control window by `Options->Execute IDL`.

3.1.3 Graphics Asynchronous Menu

This menu is started automatically. Scatterplots and plot commands are not possible.

```
lea -pro mylea -disp
```

Commands are entered in parallel to the display menu. Data input runs in a separate thread.

3.1.4 LeA Control Menu

This menu (Fig. 3.1) is started automatically. Scatterplots and plot commands are not possible.

```
lea -pro mylea -men
```

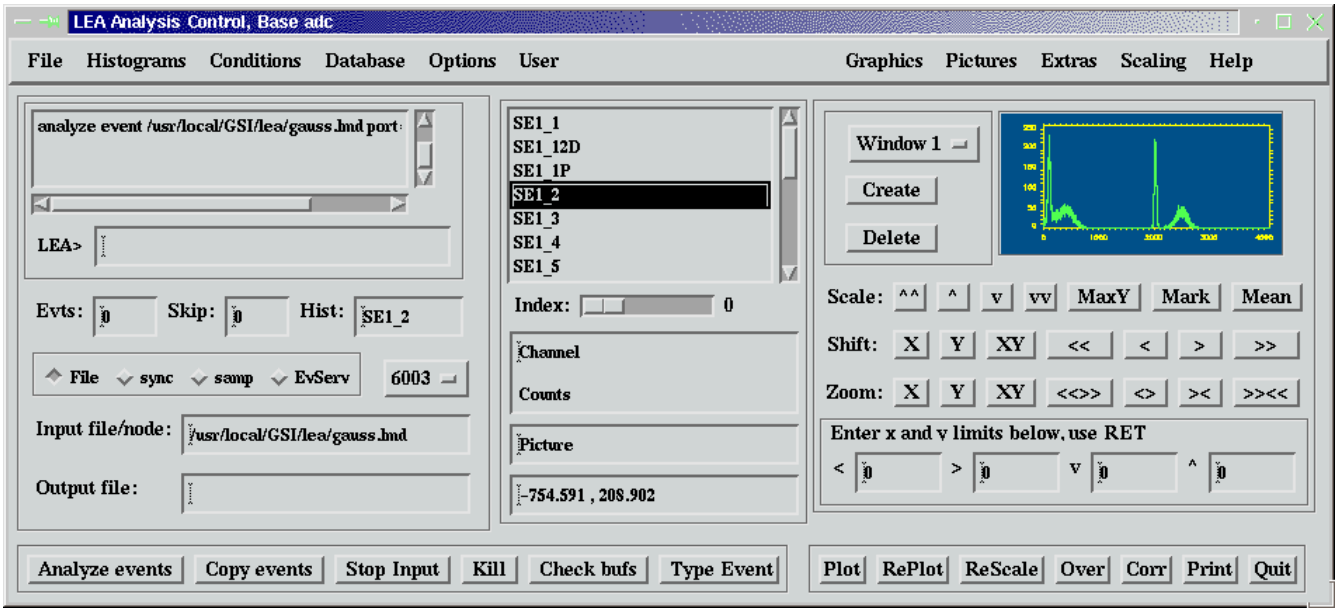

Figure 3.1: Main Control Menu

The left part controls the input, the middle shows the histogram list and the right part controls the graphics. Into the field with the default 'Picture' you may enter your picture headline.

I/O Channels

Select the input source by pressing `File`, `sync` (MBS transport), `samp` (MBS stream server) or `EvServ` (MBS event server or remote event server) and input the file or node name, respectively.

Optionally you may specify an output file name. To specify more parameters for the output, use a menu called by `File->Output` (Fig. 3.2, page 28). The buffer type/subtype should be

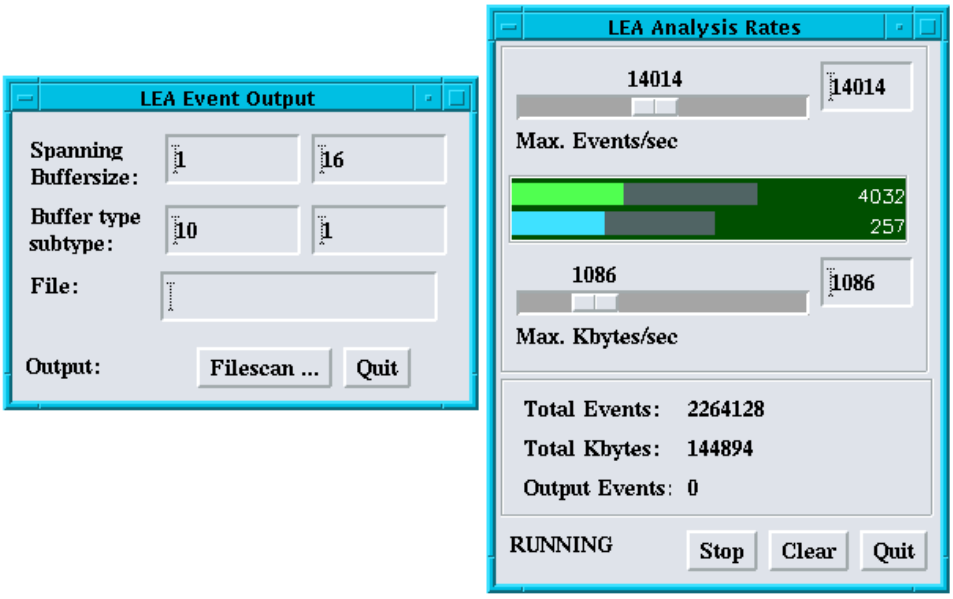

Figure 3.2: Left: output menu, right: rate meter

10,1 when the events to be written have the standard MBS format. One can select 4,1 when the events have type/subtype 4,1. The format of such events must be a longword with the data size in 16-bit words followed by two words type/subtype (4,1), and then the data. The spanning parameter specifies over how many buffers an event can span. The buffer size is in Kbytes.

Analyse Events

Then press `Analyze events` to start the input, and `Stop Input` to stop. A rate meter window pops up (Fig. 3.2, page 28).

Copy Events

Button `Copy events` copies events from input channel to output file.

Check Buffers

Button `Check bufs` checks the consistency of buffers. This command works only for `File`, `sync`, and `samp` input.

Type Event

Button `Type event` prints events. This works also when an analysis is running. The format can be changed by `Options->Event Format`.

Any input channel should be closed by `Stop input`. In case the analysis is waiting for data or looping forever, button `Kill` can be used to kill the thread.

3.2 Graphics

3.2.1 Display Profile

You may setup several display default parameters in the IDL profile. This file (`leaini_mylea.idl`) was created by `lea -new mylea`. Recommended modifications:

```
                                  | before LEAset,
print,'Your announcement'          | output at startup
!x.style=1                        | exact axis limits (0 round up)
                                  | after LEAset,
title='Your window title',$       | Control window title
queue='Your prefered printer',$   | default printer
window=[Xsize,Ysize,Xpos,Ypos],$  | display window in pixels
min=0,$                           | Do not suppress zero channels
input='xxx',$                     | Default input
/file,/sync,/sample,/evserv,$     | Type of input
```

Note that the `,$` are required, because all these lines are arguments for `LEAset` (the `$` is in IDL the line continuation character).

3.2.2 1-D Histogram Display

Figure 3.3 on page 30 shows several overlayed histograms and an additional lettering. To overlay histograms, select first one in list and `Plot`. Scale it as needed. Then select the next one, change line color (`Graphics->Lines`), and `Over`, and so on. The overlays remain until `Plot` is pressed. Zooming and scaling can be done using the arrow buttons or by cursor (see 3.2.3). A small additional menu with all widgets for zooming and shifting can be opened by `Graphics->Controls`, as shown in 3.4 on page 30. A smart expander can be opened by `Graphics->Expand` (fig. 3.5 on page 31). One can use the sliders to change the display limits or the cursor. Move cursor to a border (cursor symbol changes), press LM and move. All zooming is done in live mode. With

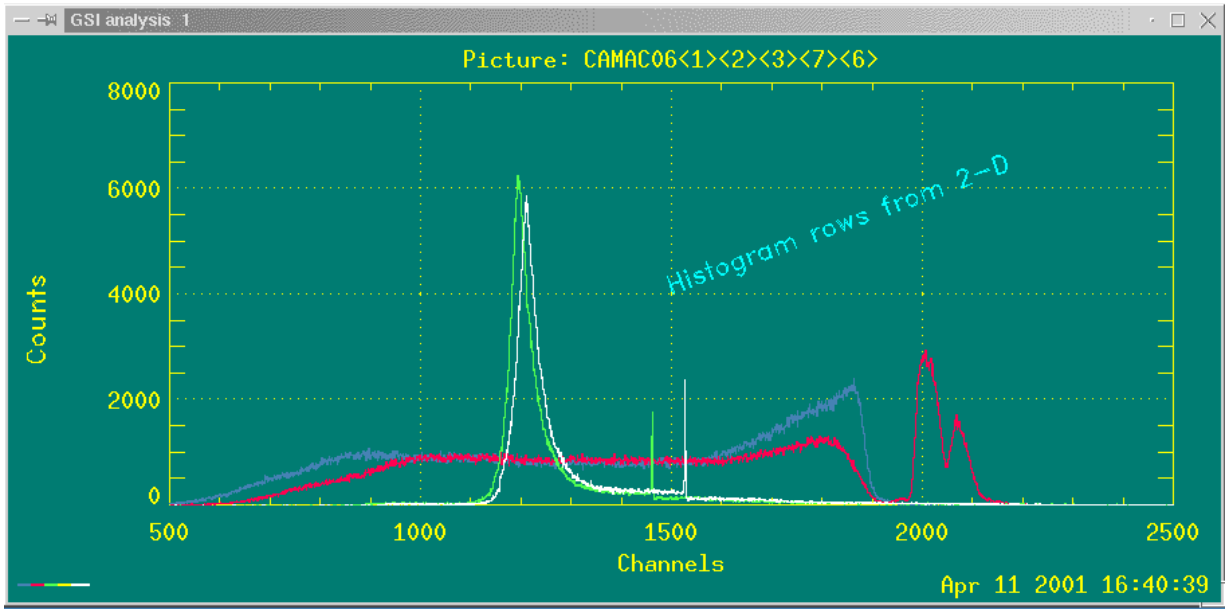

Figure 3.3: Example of histogram display

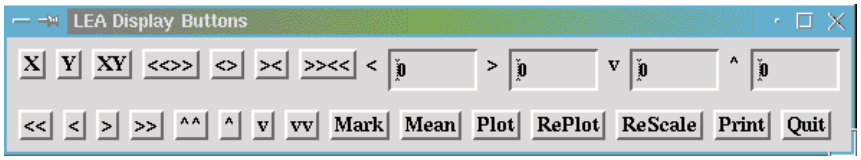

Figure 3.4: Extra Display Buttons

the cursor inside the borders, pressing LM and moving moves the borders. It is recommended to enable exact scaling by `Scaling->X Exact` and `Scaling->Y Exact`.

Binsize

The binsize for display can be set by `Graphics->Binsize`. In the preview window the current histogram is plotted with binsize 1 and the selected binsize. All numerics are independent of binsize.

3.2.3 Cursor

The cursor operates in several modes:
1. Default: cross hair. Mark a 2-d window by pressing LB at first point, move cursor and release

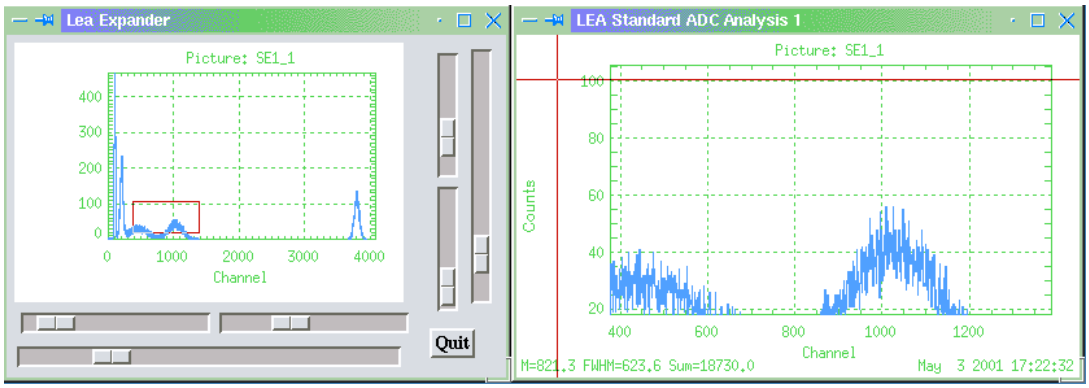

Figure 3.5: The expander with sliders and active cursor

at second point.
It is possible to use the interactively set cursor coordinates in user programs, i.e. analysis functions or commands. Call function `f_his_cursor` to receive the coordinates of the two points set by cursor:

```
f_his_cursor(&x1,&y1,&x2,&y2);
where x,y are float values
```

2. Mark: After pressing `Mark` button, pressing the LB plots the channel and histogram content vertically and adds this text element to the lettering (see below). Stop with RB.
3. Mean: After pressing `Mean` button, press LB at left, move to right limit, and release to plot mean and FWHM values horizontal and add this text to the lettering. Stop with RB. Call function `f_his_idl_get_mean` to receive mean, fwhm, and sum for each range as vectors:

```
stat = f_his_idl_get_mean(ra_peaks,ra_fwhm,ra_sum,fieldsize,&l_peaks);
where fieldsize is the length of (float) vectors ra*
stat 0: OK, -1: fieldsize too small, >0: wrong data (no fit done before)
```

The other modes are enabled by pressing `X`, `Y`, or `XY`, i.e. to expand with the cursor, first press `X`, then press LB at left limit and move to right limit.

3.2.4 Additional Display Windows

With button `Create` more graphics windows can be created. Note that the focus to a graphic window must be set by selecting the window with the pull down menu `Window n`.

3.2.5 Display Pictures

In the display profile file also pictures may be defined, normally by calling another IDL file. An example is `LEA$EXE:LEAINI_ADC.IDL ($LEA_DIR/idl/leaini_adc.idl)` which calls `LEA$EXE:PICT_ADC.IDL`

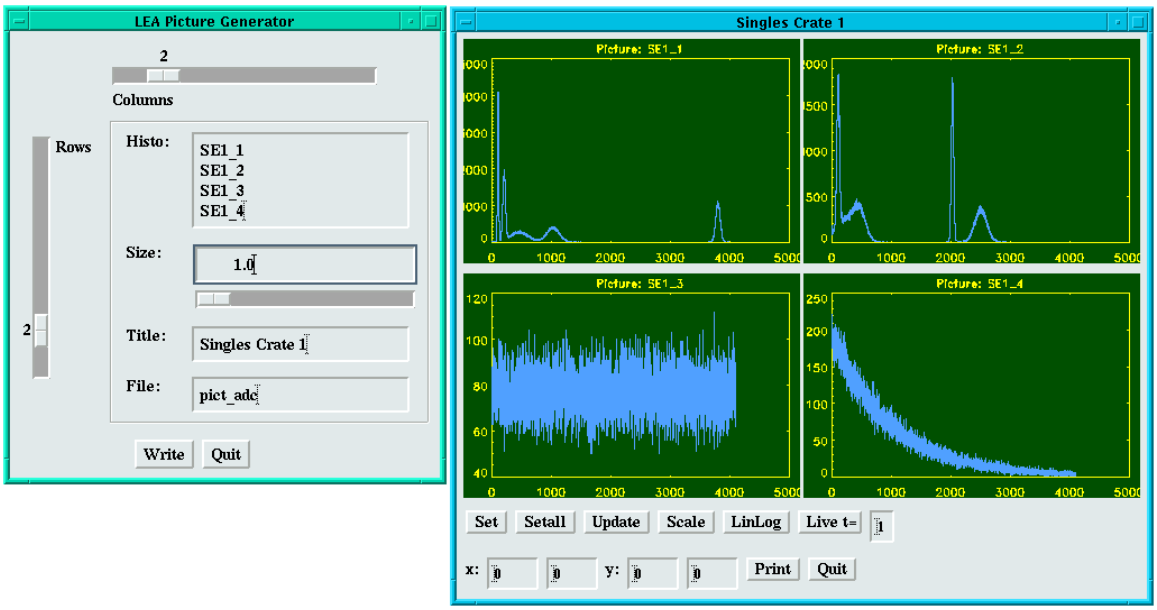

Figure 3.6: Left: picture definition, right: picture

(`$LEA_DIR/idl/pict_adc.idl`). A picture is defined by calls of `leapicdef`:

```
leapicdef,name="picture name",row=3,col=2,size=1
leapicdef,"HISTO1"
leapicdef,"HISTO2"
leapicdef,"HISTO3"
leapicdef,"HISTO4"
leapicdef,"HISTO5"
```

The first call defines the picture setup, the subsequent ones the histograms for the frames. If one has 2-dim. histograms used as indexed 1-dim histograms, these can be entered into a picture by

```
leapicdef,name="picture name",row=3,col=2,size=1
leapicdef,"HISTO1",index=1
leapicdef,"HISTO1",index=3,to=6
```

In this example the histogram row with index 2 is not in the picture. IDL routines to create pictures can be generated by a menu from the control menu by `File->Picture` (Fig. 3.6, page 32). To add histograms one can click on the histogram list or enter and edit the histogram table. With button `Write` a file `<file>.idl` is written. Once written, the picture definition routine cannot be modified through the menu, but by any text editor. Call this IDL procedure at the end in your `leaini_<project>.idl` procedure by

`@<file>.idl`

Pictures keep several frames in one window. They are displayed from the menus under `Pictures`. The pictures can be displayed in live mode, i.e. they are updated every n seconds after pressing `Live`. Stop Live mode by `Update`. It is not intended to manipulate the frames in pictures. Instead, clicking a frame this frame will be displayed in the display window and can be manipulated there. With button `Set` the limits for the frame are taken from the display window. With `Setall` all frames are set to these limits. The limits entered in the text fields are applied to all frames.

3.2.6 2D Graphics

Two-dimensional histograms can be displayed in different modes selected by a separate control panel (Fig. 3.7, page 33). Button `P1` plots all bins greater then 1 with the same color, `Point` plots

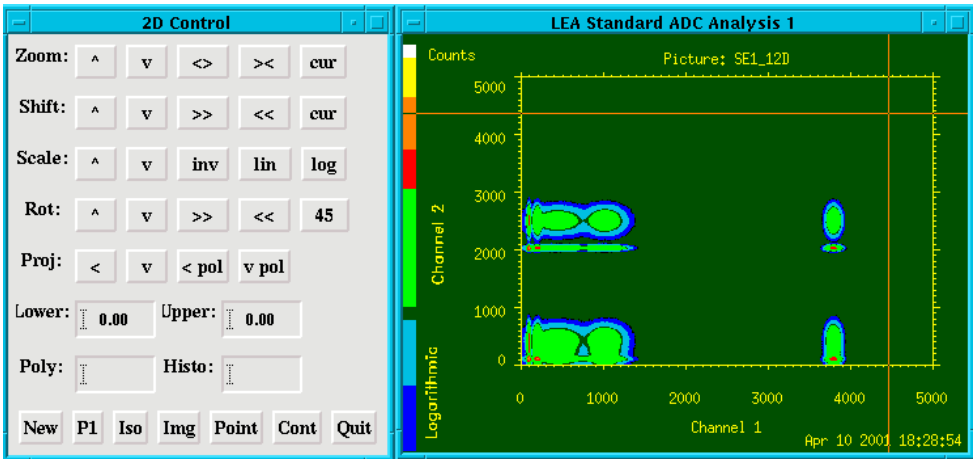

Figure 3.7: 2D Control Menu

the bins with a color scale. `Img` plots the bins as pixels and the window size needed is defined by the histogram size.

Projections

Projections into 1-dimensional histograms can be done by command `project` or from the 2D panel. One can specify limits or a polygon file. For projections from the 2D panel one **must** specify a window by cursor or values, except a polygon file is specified. The target histogram must match type, limits and number of bins of the according coordinate of the source histogram.

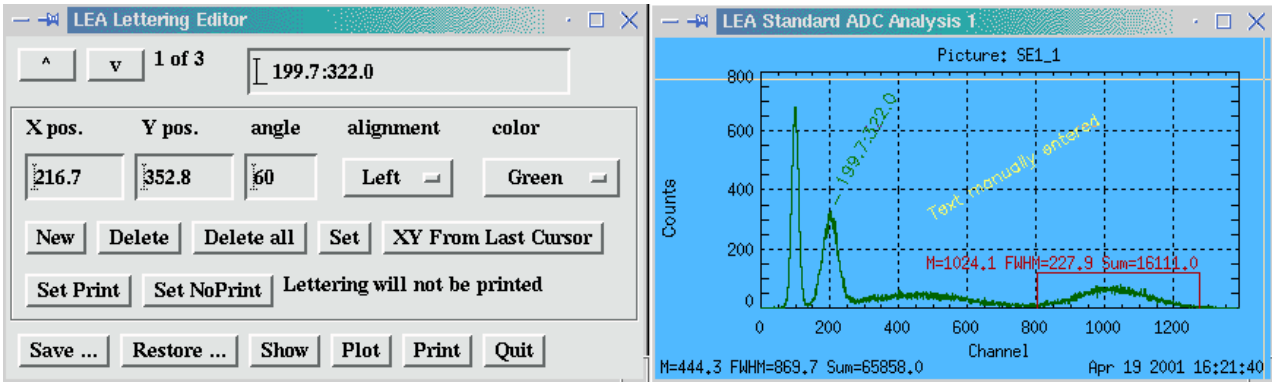

Figure 3.8: Lettering Menu

3.2.7 Adding Lettering to Canvas

There is a new menu to add or modify lettering. See fig. 3.8, page 34. It is opened by `Graphics->Lettering`. The lettering element consists of text elements with a string, the position (in data coordinates), angle, alignment and color. Elements can be modified (change values and press `Set`). To add a new element, first press `New`. To place an element with the cursor, press LB on the position in the canvas and then press `XY From Last Cursor`. The position fields will be updated, but the change takes effect only after pressing `Set`. With `Plot`, the lettering is plotted into the current display canvas. By default, the lettering will not be printed. If it shall be printed, press `Set Print`. Note that some cursor operations add text elements independent if the lettering menu is up, i.e. in `Mark` or `Mean` mode. A lettering can be saved/restored to/from files.

3.2.8 Polygons

Polygons are stored in text files or polygon conditions. They can be created and edited by a polygon editor. Normally one displays a 2-dimensional histogram or scatter plot, and calls the editor by `Extras->Polygon`. One can enter a new polygone with button `New`, or first restore a polygon from a file with the `Restore` button. One can add, delete and move points. The cursor symbol changes when a click will catch a valid point. When the polgon is OK, save it to a file with `Save`. The polygon edit panel is shown in Fig. 3.9, page 35.

Plot Polygon

A polygon is plotted in a scatterplot frame by

```
lea> plot polygon <file> color=c
```

where <`file`> is generated by `Exp data` in the condition menu or `Save` in the polygon editor.

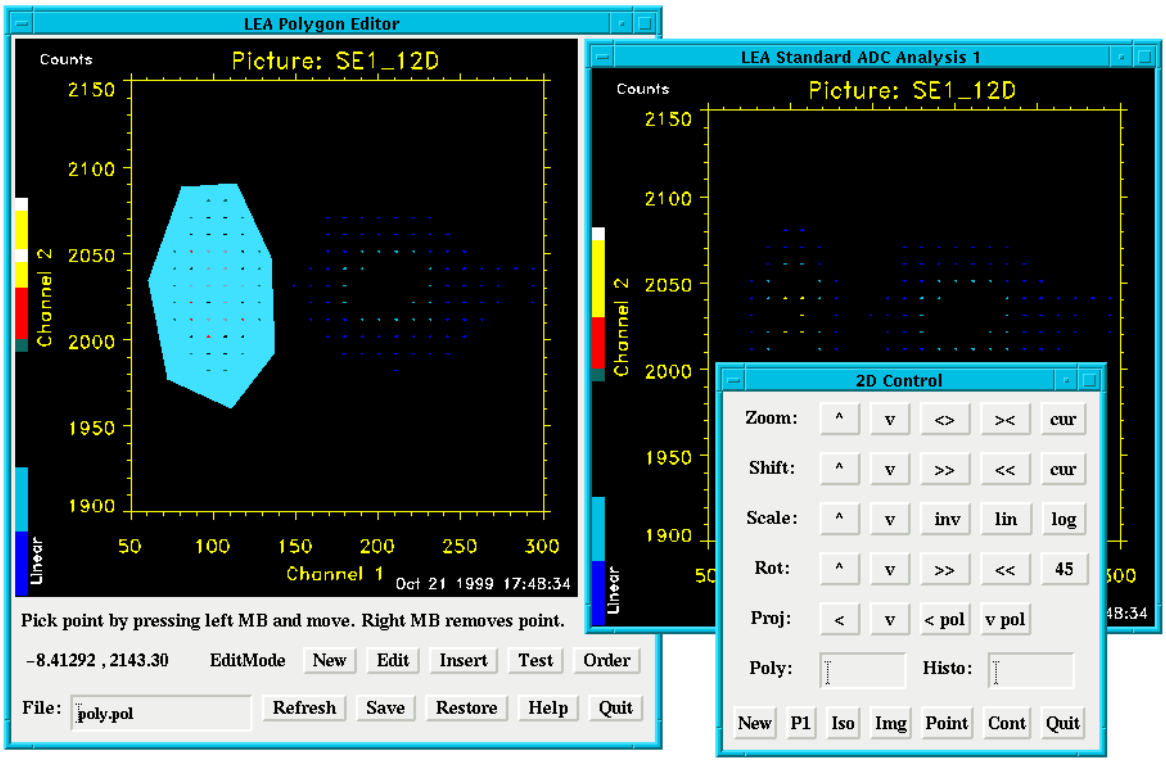

Figure 3.9: Polygon Editor

3.2.9 Conditions

For managing window and polygon conditions an extra GUI is provided as shown in Fig. 3.10, page 36. It is called from the menubar of the main window. There are buttons specific to the window or polygon conditions besides the namelists. The buttons at the bottom are common and apply to the last selected list item, or condition type, respectively. The `Include` buttons write definition files for the conditions. The names are `windef_<name>.h` or `poldef_<name>.h`, where <name> comes from the `Name` field. The `FixTrue - FixFalse - Check` selector (un)freezes the selected condition, the `Clear` button clears the test and true counters, the `Graph` button prints a primitive bar chart, the `Dump` button writes conditions (window or polygon) to a file `<name>.win` or `<name>.pol`, respectively. Therefore do not use dots in the `Name` field.

Window Conditions

Window limits can be set numerically by the limits fields (`From values`) or by cursor: press LB on the first point, move, and release mouse at the second point. Then press `From Cursor` to copy the values to the selected window condition. The `Display` button displays the selected window

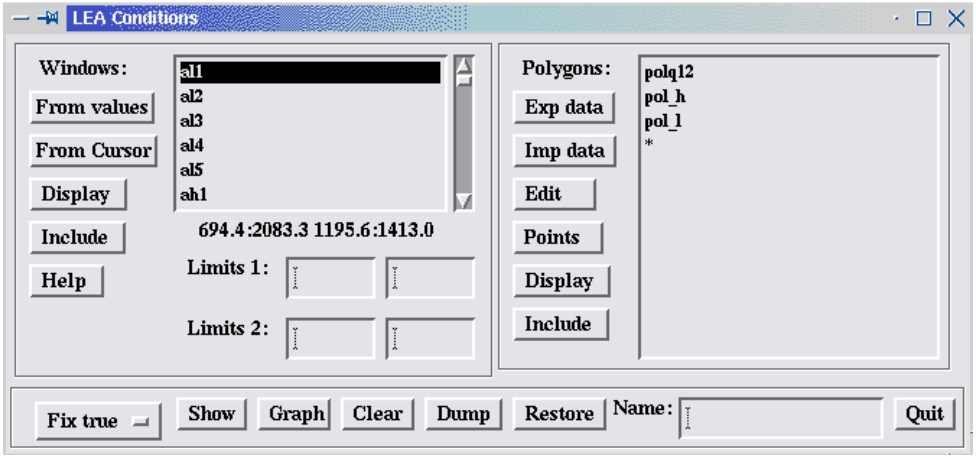

Figure 3.10: Condition Menu

overwriting cursor values previously set.

Polygon Conditions

At the polygon side the `Exp data` button writes the data of the selected polygon to a file. If the `Name` field is empty, the polygon name is used. A `.pold` is appended any case. These files can be used by the polygon editor and for projections. The `Points` button lists the selected polygon and all values. The `Edit` button starts the polygon editor which is the same as described above, but running in a different mode as shown in Fig. 3.11, page 37. It displays immediately the selected polygon. Note the window header showing the name of the polygon (p2 in Fig. 3.11). There is also an additional button right of the `Save` button with the polygon name. Pressing this button, the actual polygon is copied into the condition polygon. The `Save` button would write the polygon to the file like `Exp data` in the condition menu.

3.2.10 Plot Command

Once the display is started, histograms can be displayed from LeA by command `plot histogram`. This command can be exectued only, if no menu is active.

3.2.11 Fitting

There is a simple tool for fitting Gauss shaped peaks in histograms. Fit parameters are setup in a little menu called by `Extras->Fit Setup` shown in fig. 3.12, page 38. If there are two or more peaks a background can optionally subtracted. The backround is fitted as polynomial (maximum order 2) to the reagions between the peaks. The peaks are searched. The minimum required

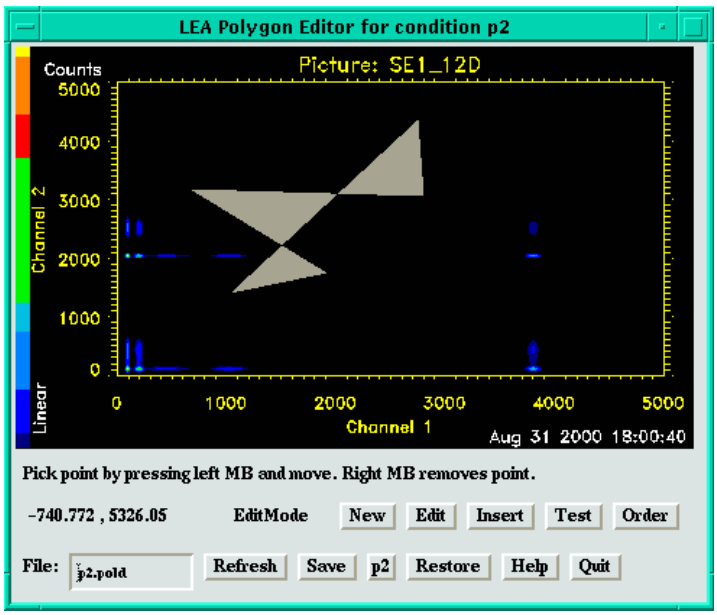

Figure 3.11: Polygon Condition Editor

distance between a peak and a valley can be adjusted by two values: first in % of the current channel, second as histogram-maximum/value. If the percent value is too high, peaks will be missed, if it is too low, fake peaks will be found. If value n is too small, peaks will be missed, if it is too big, fake peaks will be found. The minimum distance avoids fake peaks at the bottom of a histogram.

Fitting can be done in the display menu, by L^{e}A command `fit histogram` or by L^{e}A command `plot histogram .. -FIT`. The fit results can be obtained in user programs, i.e. analysis functions or commands. Call function `f_his_idl_get_fit` to receive mean, max, fwhm, and sigma as vectors:

```
stat = f_his_idl_get_fit(ra_peaks,ra_max,ra_fwhm,ra_sig,fieldsize,&l_peaks);
where fieldsize is the length of (float) vectors ra*
stat 0: OK, -1: fieldsize too small, >0: wrong data (no fit done before)
```

3.2.12 Scatterplots

Scatterplots are possible only, if no menu is active. A display frame for scatter points can be created by command `scatter`. IDL must be started. The command specifies the limits and letterings. Then in the analysis routine one may call routine `f_his_idl_scat(x,y,c,p)` to plot one point. The arguments are:

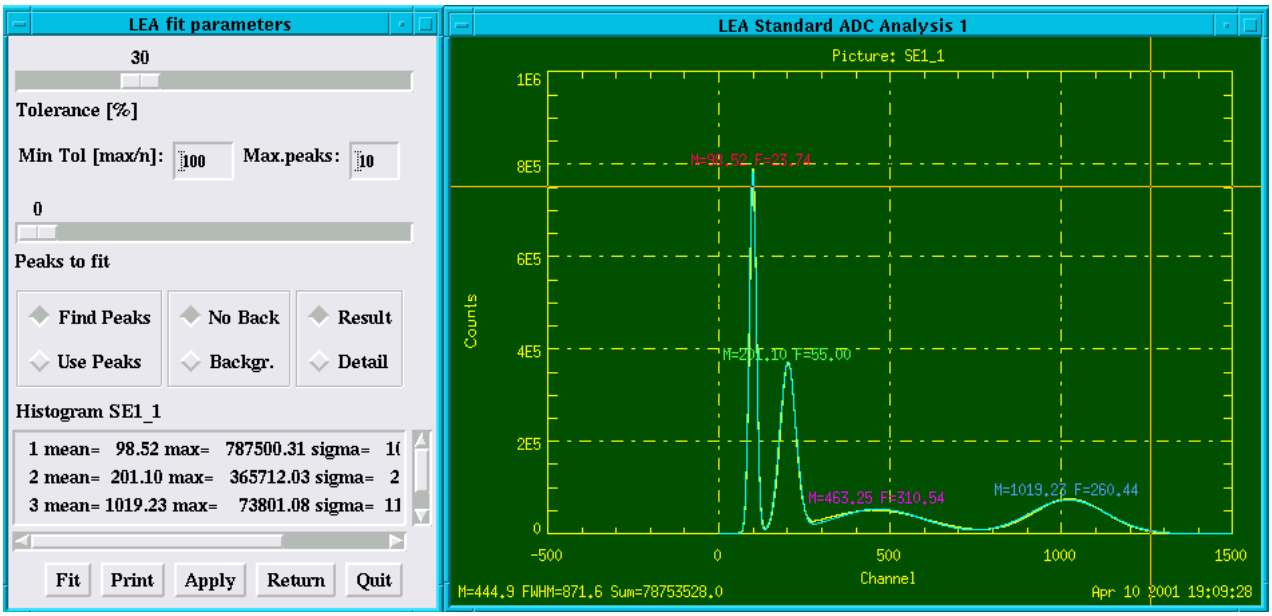

Figure 3.12: Fit Menu and Result

```
x     float value X
y     float value Y
c     color = SCAT__DEFAULT
              SCAT__RED
              SCAT__BLUE
              SCAT__GREEN
              SCAT__YELLOW
              SCAT__WHITE
              SCAT__CYAN
              SCAT__MAGENTA
p     Points per buffer.
```

If color is `SCAT__DEFAULT`, all points are plotted in the current color. This is little faster than plotting each point with a different color. Output of scatter points can be stopped by command `scatter -OFF`. When a scatterplot is finished one can get a window copy by command `plot image`, or after starting the display menu (`disp`) by `File->Image`. Enter the file name and press the desired format (gif, ps, bmp).

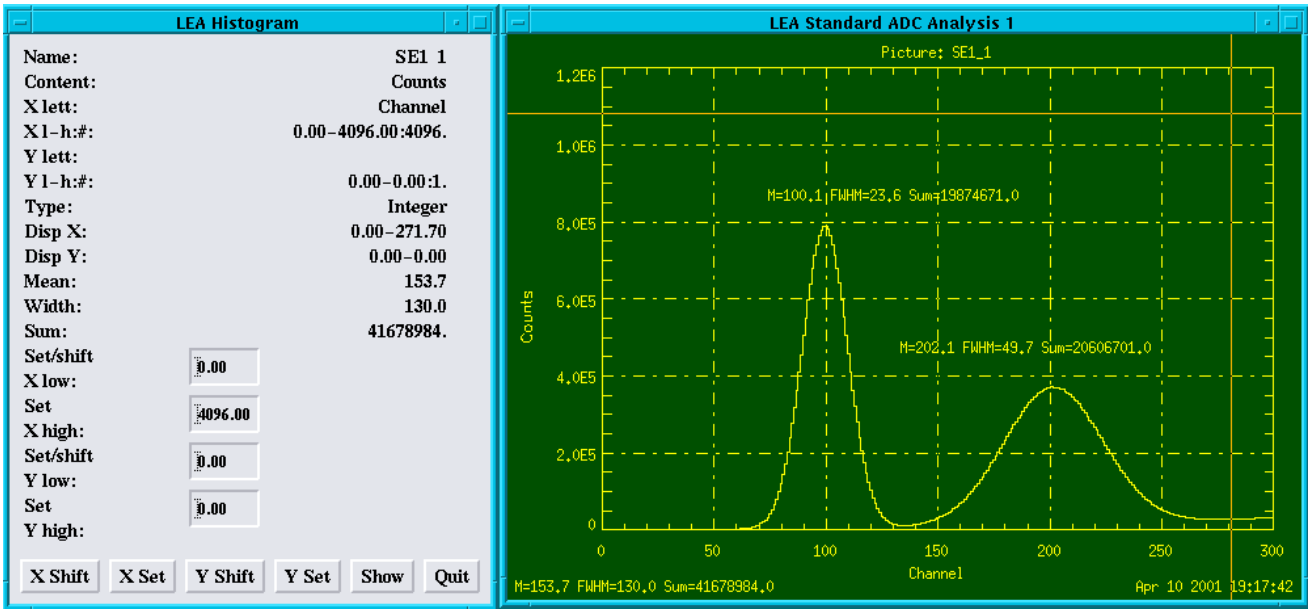

Figure 3.13: Histogram Header Information

3.3 Histogramming

3.3.1 Header

There is now an optional window displaying information about the current histogram. The window is opened by `Histogram->Header`. See fig. 3.13, page 39

3.3.2 Changing Limits

Entering values into the input fields and press `X Set` or `Y Set`, respectively, one can change the histogram calibration on the fly. The number of channels remains, but the binsize changes. Data already collected in the histogram will look the same, but newly accumulated data will reflect the new boundaries.

3.3.3 Calculations

There is a menu to calculate histograms from others (Fig. 3.14, page 40). Histograms must have same type and size. Multiply and Divide are provided only for float histograms. For subtraction the histogram to be subtracted (2nd) can be optionally scaled, either by a value or to achieve the same maximum as histogram 1.

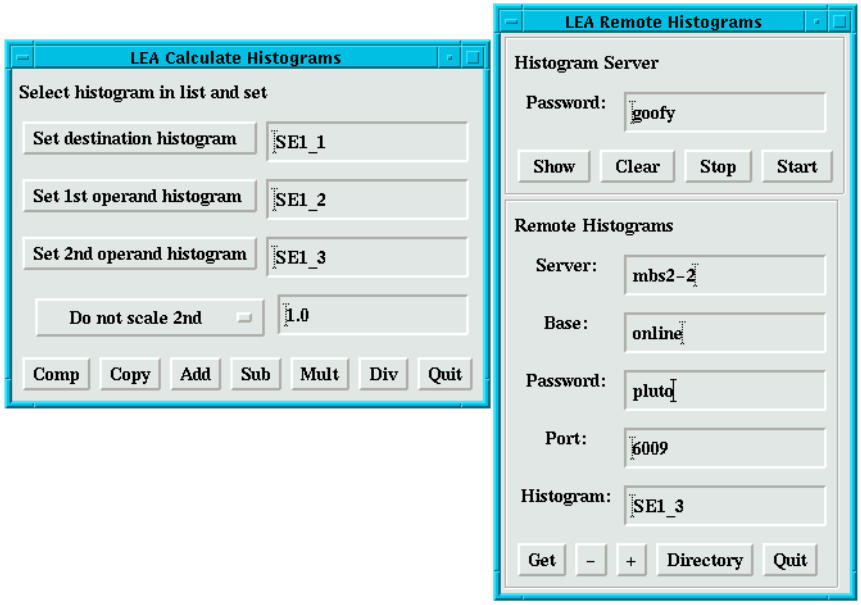

Figure 3.14: Left: histogram calculations, right: histogram client/server

3.3.4 Server

A histogram server can be started by menu `Histogram->Remote` (upper part in fig. 3.14, page 40). Optionally a password can be specified which than must be given by any client. Note which port the server actually takes. There is an Origin client available.

3.3.5 Client

The lower part of the menu allows to get histograms from a server. The histograms must exist and match in type and size with the remote ones. The directory button lists these histograms.

3.4 Save and Restore Data Base

Histograms and conditions are kept in the data base. The data base can be saved to file by menu `Database->Save`. It is recommended to save the base in compressed format. If selecting to write a new file, an existing file is not overwritten. To overwrite an existing file, use replace. The data base can be restored, but only if the structure matches exactly the current data base.

3.4.1 Restoring during Startup

Instead of creating a data base in the startup procedure `leaini_<project>.scom` (bascre... and hiscre...) one can restore a data base from file by

```
restore base <name> <file> -new
```

3.4.2 Autosave

The autosave mechanism saves the data base every n seconds. To enable autosave, enter command (or put it in the startup file)

```
autosave base <file> seconds -on [-print]
```

A minimum of 10 seconds (is also default) can be specified. After the given time, the analysis loop is informed to dump the histogram base in compressed mode. Therefore the dump is always consistent. If the analysis thread is not running or is waiting for data, no dump is done. If the analysis is stopped, and autosave is active, a dump will be done immediately. The optional -PRINT switch activates a message for each dump.

- `autosave base -off` : stop autosave
- `autosave base` : disables print output
- `autosave base -PRINT` : enables print again.
- `autosave base sec=s` : sets a new intervall (and disables output)
- `autosave base -SHOW` : outputs the status

When autosave is activated, the "dump base" command is deactivated.
The rate meter shows the count down for the next dump.

3.4.3 Radware Histograms

Export histograms in Radware format by command:

```
lea> dump histogram <name> <prefix> -radware [-over]
```

The command dumps histograms specified by <name> one by one to files named <prefix>_<histogram>.spe. Without `-over` files are not overwritten. Histograms are converted to float. 2-d histograms (integer only) must have 4096 x-channels and are stored as Radware matrix (.mat).

On the GSI central Linux machines the Radware package is enabled by

```
. radwarelogin
```

Then RADWARE_HOME points to the radware top directory. $RADWARE_HOME/doc has some docu. One can directly call the radware programs like:

```
gf3
```

3.5 Direct Event Access

There is a mechanism to access events randomly in an lmd file. To use it, one has to create a tagfile from the lmd file.

```
lea> create tag file <lmd file> [<tag file>]
```

If <tag file> is omitted, <lmd file> with postfix .lmdir is used, i.e. if lmd file name is abc.lmd:

```
create tag file abc      ! creates abc.lmdir
create tag file abc.lmd  ! creates abc.lmdir
create tag file abc abc1 ! creates abc1.lmdir
```

Three commands got new arguments to use tag files:

```
type/copy/analyse event
    tagfile=<name>
    number=<event number>
    sample=<skip events>
    -INDEX
```

"number" is the real event number or the event index in tag file when -INDEX is given. "sample" is the number of events to skip before process next event. By this one can scan a large lmd file fast analysing only a fraction of all events. When argument "number" is specified, the tagfile name defaults to the lmd file name with postfix .lmdir. When no tagfile is found the standard event IO functions are used. When argument "number" is NOT specified standard event IO functions are used.

The mechanism is very fast to access events in large files, i.e. get 1% of events of a file or skipping a large number of events.

Chapter 4

LEA Sources

GMC_LEA_SRC

4.1 Files and Modules

4.1.1 Directories

VMS

```
LEA$SRC:    C sources, test data
LEA$INC:    h files
LEA$EXE:    programs, DCL procedures, LEA procedures and IDL procedures
LEA$LIB:    libraries, IDL sources, help files, command definition
```

AIX and Linux

```
$LEA_DIR              Makefile, LEA procedures, command definition
$LEA_DIR/src          C sources
$LEA_DIR/inc          h files
$LEA_DIR/idl          IDL sources and procedures
$LEA_DIR/doc          Postscript manuals and help files
$LEA_DIR/examples     Example project and data file
$LEA_DIR/$GSI_OS/bin  programs and scripts
$LEA_DIR/$GSI_OS/lib  libraries
$LEA_DIR/$GSI_OS/obj  objects
```

The main program is `m_lea.c`. The histogramming functions are in files `f_his*.c`. The command interface is `f_cmd.c`. The event routines are `f_evt.c`. The TCP library is `f_stccomm.c`. The user analysis routine is `f_anal`. It must have an initialization entry `f_anal_init`. On VMS the IO routines `i$opolf` and `i$opilf` are used.

4.1.2 Macro HISDEFt

This macro is called in the automatically created `hisdef_<project>.h` include file. The `t` stands for `I` (integer histogram) or `R` (floating point histogram). These macros must be called outside `f_anal_init` and `f_anal`. The argument is the histogram name. It declares pointers `ps_spe_name` and `pl_name` or `pr_name` to the histogram header (structure s_spe), and the data array, respectively. Example:

```
HISDEFI(SP1) /* defines static s_spe *ps_spe_SP1 and static int   *pl_SP1 */
HISDEFR(SP2) /* defines static s_spe *ps_spe_SP2 and static float *pr_SP1 */
```

4.1.3 Macro HISINIt

This macro is called in the automatically created `hisini_<project>.h` include file. The `t` stands for `I` (integer histogram) or `R` (floating point histogram). These macros must be called in `f_anal_init` when the argument pointer is not `NULL` (which means that `f_anal_init` has been called by the `initialize` command). The argument is the histogram name. It sets the pointers declared before in the `HISDEFt` macros. The macro returns a status in `l_status` which must be declared as `int`. It also needs the data base pointer `ps_head` which is declared in `f_anal.h` which must be included outside `f_anal_init` and `f_anal`. Example:

```
HISINII(SP1)  /* sets ps_spe_SP1 and pl_SP1 */
HISINIR(SP2)  /* sets ps_spe_SP2 and pr_SP2 */
```

4.2 Installing LeA

These instructions are also in `AnInstallationGuide.txt`.

LeA comes as zipped tar file. The file name is built from the version, e.g. leav26.zip for LeA version 2.6.

4.2.1 Unpacking

We recommend to have an account for managing LeA . The top directory for LeA should be xxxx/lea, e.g. `/u/leaman/lea`. Copy the zip file here, unzip and untar it:

```
cd /u/leaman/lea
unzip leav26.zip leav26.tar
tar xf leav26.tar
```

creates top directory `/u/leaman/lea/v26`.

4.2.2 Settings

Variables needed from outside: `GSI_OS` must be Linux or AIX. It is used to select the binary directories. cd to the top directory. There are two files you must first adjust:

idllogin

Defines `IDL_DIR, LM_LICENSE_FILE, IDL_PATH`, and for Linux `LD_LIBRARY_PATH`.

lealogin

Defines `LEA_DIR` as top directory, e.g. `/u/leaman/lea/v26` and adds `$LEA_DIR/idl` to `IDL_PATH`. Uncomment the `HELPDIR` definition. Make `idllogin` and `lealogin` accessible by creating links, i.e. on /usr/local/bin.

4.2.3 Build LeA

cd to `$LEA_DIR` (if you are not still there) and execute

```
. idllogin
. lealogin
printenv|grep IDL
IDL_DIR=
/usr/local/lpp/idl.53/idl_5.3
IDL_PATH=
/u/leaman/lea/v26/idl:
/usr/local/lpp/idl.53/idl_5.3/lib:
/usr/local/lpp/idl.53/idl_5.3/examples:
/usr/local/lpp/idl.53/idl_5.3/lib/obsolete
printenv|grep LEA
LEA_DIR=
/u/leaman/lea/v26
```

Now build LEA by

```
gmake
```

Eventually one has to set the execution bits of the scripts:

```
chmod 777 $LEA_DIR/$GSI_OS/bin/*.sc
```

4.2.4 Make Compiled IDL Image File

You are still on `$LEA_DIR`; cd to subdirectory `idl` and start prebuilt program `m_lea_deve` by

```
../$GSI_OS/bin/m_lea_deve -s=../leaini_adc -i=leaini_adc -menu
```

A long list of compile messages should be displayed and the LeA menu starts up. On the left side enter in the file/node text field:

```
../examples/gauss<RET>
```

Press button <Analyze events>. The rate meter should pop up. Click on first histogram of the list and press button <Plot>. The histogram is displayed. Do a fit from menu bar Extras->Gauss Fit. Some compile messages should be displayed. Call from menu bar Colors->ColorTable. Some compile messages should be displayed. Now all IDL modules needed should be compiled. Save the IDL image from menu bar File->xdrSave to file `leaidl.sav`. Quit **L^{e}A** and move `leaidl.sav` to `../$GSI_OS/lib`.

4.2.5 Testing

Now one should login under a user account, execute the `. idllogin` and `. lealogin` scripts and then

```
lea -fun -men
```

Analyze the same file as above.

Chapter 5

LEA Command Summary

ANALYZE EVENTS input [events skipevts skipbufs file size type subtype span port number tagfile sample]-TRANS -STREAM -EVENT -REVSERV -FILE -ALL -HEADER -WAIT -COMPRESS -INDEX
(m_lea) Read and analyze events.

ATTACH BASE name
(m_lea) Attach existing data segment for histogram manager.

AUTOSAVE BASE file seconds -ON -OFF -PRINT -SHOW
(m_lea) Dump base into file, for histogram manager.

CALCULATE HISTOGRAM destination 1source [2source factor]
-COMPARE -COPY -ADD -SUBTRACT -MULTIPLY -DIVIDE -SCALE
(m_lea) Calculate histogram in histogram manager.

CLEAR CONDITION name
(m_lea) Clear window condition counters.

CLEAR HISTOGRAM name -LOG
(m_lea) Clear histogram in histogram manager.

CLEAR POLYGON name
(m_lea) Clear polygon condition counters.

CLEAR WINDOW name
(m_lea) Clear window condition counters.

COPY BUFFERS input [file buffers skip]-TRANS -STREAM -FILE -ALL -WAIT -CHECK -COMPRESS

(m_lea) Copy buffers to file.

COPY EVENTS input [file events skipevts skipbufs size_kb type subtype span port number tagfile sample]
-TRANS -STREAM -EVENT -REVSERV -FILE -ALL -WAIT -COMPRESS -INDEX
(m_lea) Copy events to file.

CREATE BASE name histogram size
(m_lea) Create data segment for histogram manager.

CREATE DIRECTORY POLYGON polygons -NEW
(m_lea) Initialize polygon conditions.

CREATE DIRECTORY WINDOW windows -NEW
(m_lea) Initialize window conditions.

CREATE FILES project [-ANALYSIS -STARTUP -CREATE -MBS -WINDOWS -POLYGONS]
(m_lea) Create include files and procedure.

CREATE HISTOGRAM name dim type 1channels 1low 1up [1binsize 2channels 2low 2up 2binsize]
(m_lea) Create histogram for histogram manager.

CREATE POLYGON name [description datafile]
(m_lea) Create polygon condition.

CREATE TAG FILES lmdfile tagfile
(m_lea) Create tag file from lmd file.

CREATE WINDOW name [description x1 x2 y1 y2]
(m_lea) Create window condition.

DELETE BASE [name]
(m_lea) Delete data segment for histogram manager.

DELETE HISTOGRAM name
(m_lea) Delete histogram in histogram manager.

DELETE POLYGON name
(m_lea) Delete polygon condition.

DELETE WINDOW name
(m_lea) Delete window condition.

DETACH BASE [name]
(m_lea) Detach data segment for histogram manager.

DISPLAY [command][histograms][exclude]-START -STOP -LOOP -MENU -SOURCE -PROJECT
(m_lea) Enter IDL.

DUMP BASE base file -COMPRESS -OVERWRITE
(m_lea) Dump base into file, for histogram manager.

DUMP HISTOGRAM name file -SEPARATE -EXPORT -RADWARE -OVER
(m_lea) Dump histogram in file or files.

DUMP POLYGON file
(m_lea) Dump all polygon conditions.

DUMP WINDOW file
(m_lea) Dump window conditions.

EXPORT POLYGON DATA name [file]
(m_lea) Dump one polygon for edit.

EXPORT WINDOW DATA name
(m_lea) Export window condition for display.

FIT HISTOGRAM name [peaks tolerance]-BACK
(m_lea) Display histogram.

FREEZE CONDITION name -TRUE -FALSE -CHECK -UNFREEZE
(m_lea) (un)freeze condition execution.

FREEZE POLYGON name -TRUE -FALSE -CHECK -UNFREEZE
(m_lea) (un)freeze condition execution.

FREEZE WINDOW name -TRUE -FALSE -CHECK -UNFREEZE
(m_lea) (un)freeze condition execution.

GET REMOTE DIRECTORY server base [histogram][port][password]-VERBOSE -MATCH
(m_lea) Get histogram directory from server.

GET REMOTE HISTOGRAM server base histogram [port][password]-SUBTRACT -ADD
(m_lea) Get histogram from server.

HELP [key1 key2 key3 key4 key5]
(m_lea) Help from lea.hlp.

IMPORT POLYGON DATA name [file]
(m_lea) Restore polygon condition from file.

INTEGRATE name [1lower][1upper][2lower][2upper][polygon]-FILE
(m_lea) Integratehistogram.

PLOT HISTOGRAM name
[xlower xupper ylower yupper ymin xaxis yaxis title queue file color peaks tolerance index]
-BACK -FIT -LOG -SCALE -OVER -PRINT -FIT
(m_lea) Display histogram.

PLOT IMAGE file -GIF -BMP -PS -JPEG -TIFF
(m_lea) Write image file of current display window.

PLOT POLYGON [polygon][file][color]
(m_lea) Display polygon.

PROJECT name projection [lower upper polygon]-FILE -X -Y
(m_lea) Project 2D-histogram.

PROTECT HISTOGRAM name -UNPROTECT
(m_lea) [Un]protect histogram from beeing cleared.

RATE sec -STOP
(m_lea) Shows event counter.

RESTORE BASE base file -OVERWRITE -NEW
(m_lea) Restore base from file, for histogram manager.

RESTORE HISTOGRAM name file
(m_lea) Restore histogram from file.

RESTORE POLYGON file -NEW -OVER -APPEND
(m_lea) Restore polygon condition from file.

RESTORE WINDOW file -NEW -OVER -APPEND
(m_lea) Restore window condition from file.

SCATTER [xlower xupper ylower yupper xaxis yaxis title]-OFF
(m_lea) Dsiplay scatter frame.

SET HISTOGRAM CHANNEL name value xchan [ychan]-INCREMENT
(m_lea) Set channel content of histogram in task m_lea.

SET HISTOGRAM LIMITS name [1low 1up 2low 2up]-SHIFT
(m_lea) Set limits of histogram in task m_lea.

SET HISTOGRAM SERVER -CLEAR -VERBOSE -NOVERBOSE
(m_lea) Set and Show histogram server.

SET HISTOGRAM TEXT name [text]-TITLE -XTXT -YTXT -CONT
(m_lea) Lettering text field of histogram in task m_lea.

SET VERBOSE HISTOGRAM -ON -OFF
(m_lea) Set verbosity for m_lea.

SET WINDOW name [x1 x2 y1 y2]-CURSOR
(m_lea) Set limits of window condition.

SHOW BASE [name]-FULL
(m_lea) Show information on data segment for histogram manager.

SHOW CONDITION name -DATA -FULL -GRAPH
(m_lea) Show polygon conditions.

SHOW HISTOGRAM name [1offset][1channels][2offset][2channels]-DATA -FULL
(m_lea) Show histogram info and content.

SHOW POLYGON name -DATA -FULL -GRAPH
(m_lea) Show polygon conditions.

SHOW WINDOW name -FULL -GRAPH
(m_lea) Show window conditions.

START HISTOGRAM SERVER [password]-VERBOSE
(m_lea) Start histogram server.

STOP HISTOGRAM SERVER -CLEAR
(m_lea) Stop histogram server.

STOP THREAD -KILL
(m_lea) Stop running thread.

TYPE BUFFERS input [events skipbufs subevid port]-TRANS -STREAM -FILE -SAMPLE -VERBOSE -DATA
(m_lea) Output events as in MBS.

TYPE EVENTS input [events skipevts skipbufs number tagfile sample subevid port]
-TRANS -STREAM -EVENT -REVSERV -FILE -LONG -DATA -HEX -WAIT -INDEX
(m_lea) Output buffers.

TYPE FILES input [skip blocks]-HEAD -SWAP -ASCII
(m_lea) Output hex dump of file.

Contents

1 LEA Introduction **3**
1.1 Information about LEA . . . 3
1.1.1 Web Site . . . 3
1.1.2 Mailing List . . . 3
1.1.3 LEA News . . . 4
1.2 Functionality . . . 4
1.2.1 LEA in MBS . . . 4
1.2.2 User Interface . . . 4
Line Mode . . . 4
Window Modes . . . 5
User Specific Commands . . . 5
1.2.3 Histogramming . . . 5
1.2.4 Conditions . . . 5
1.2.5 Data I/O . . . 5
Type Events . . . 6
Type Buffer . . . 6
Copy Events . . . 6
Copy Buffers . . . 6
Analyze Events . . . 6
Type File . . . 6

2 Getting Started **7**
2.1 Preliminaries . . . 7
2.1.1 Help . . . 7
2.2 Standard LEA Programs . . . 7
2.2.1 Histogramming . . . 7
Samples . . . 8
2.2.2 Special Features . . . 8
Scatter Plot . . . 8
Polygon Condition . . . 9
Tuple Mode . . . 10

2.2.3 Histogram Client . . . 10
2.3 Writing an Analysis . . . 10
2.3.1 Create Project Files . . . 10
2.3.2 Develop on OpenVMS . . . 12
2.3.3 Develop on AIX and Linux . . . 12
2.3.4 The Analysis Function (f_anal) . . . 12
Histogram Access . . . 13
Macro HISACCnt . . . 13
Macro HISSETnt . . . 13
Macro HISCLRt . . . 13
Histogram Arrays . . . 14
2.3.5 Conditions . . . 14
Window Conditions . . . 15
Polygon Conditions . . . 15
Conditions Checking . . . 15
Condition Macros . . . 16
2.3.6 Event Structures . . . 16
Byte Order . . . 16
Event . . . 17
Subevent . . . 17
GSI File Header Buffer . . . 18
GSI Buffer . . . 18
2.3.7 Creating Commands . . . 19
2.4 Starting a LeA Program . . . 20
2.4.1 Include/Exclude Histograms from GUI . . . 21
2.4.2 LeA Startup Steps . . . 21
f_anal First Initialization . . . 21
Start Procedure . . . 21
f_anal Second Initialization . . . 21
Menues . . . 21
2.5 Event Loop . . . 22
2.6 LeA in MBS . . . 22
2.6.1 Writing an Analysis in MBS . . . 22

3 Operate LEA **25**
3.1 Menu Interfaces . . . 25
3.1.1 Graphics Synchronous Menu . . . 25
3.1.2 User Graphics . . . 26
Interactive . . . 26
Callable . . . 26
User Control Menues . . . 26
Exeuting IDL Commands . . . 27

3.1.3 Graphics Asynchronous Menu . . . 27
3.1.4 LeA Control Menu . . . 27
I/O Channels . . . 28
Analyse Events . . . 28
Copy Events . . . 28
Check Buffers . . . 29
Type Event . . . 29
3.2 Graphics . . . 29
3.2.1 Display Profile . . . 29
3.2.2 1-D Histogram Display . . . 29
Binsize . . . 30
3.2.3 Cursor . . . 30
3.2.4 Additional Display Windows . . . 31
3.2.5 Display Pictures . . . 31
3.2.6 2D Graphics . . . 33
Projections . . . 33
3.2.7 Adding Lettering to Canvas . . . 34
3.2.8 Polygons . . . 34
Plot Polygon . . . 34
3.2.9 Conditions . . . 35
Window Conditions . . . 35
Polygon Conditions . . . 36
3.2.10 Plot Command . . . 36
3.2.11 Fitting . . . 36
3.2.12 Scatterplots . . . 37
3.3 Histogramming . . . 39
3.3.1 Header . . . 39
3.3.2 Changing Limits . . . 39
3.3.3 Calculations . . . 39
3.3.4 Server . . . 40
3.3.5 Client . . . 40
3.4 Save and Restore Data Base . . . 40
3.4.1 Restoring during Startup . . . 41
3.4.2 Autosave . . . 41
3.4.3 Radware Histograms . . . 41
3.5 Direct Event Access . . . 42

4 LEA Sources 43
4.1 Files and Modules . . . 43
4.1.1 Directories . . . 43
VMS . . . 43
AIX and Linux . . . 43

4.1.2 Macro HISDEFt 44
4.1.3 Macro HISINIt 44
4.2 Installing LeA 44
4.2.1 Unpacking 44
4.2.2 Settings 44
idllogin 45
lealogin 45
4.2.3 Build LeA 45
4.2.4 Make Compiled IDL Image File 45
4.2.5 Testing 46

5 LEA Command Summary **47**